

¿Qué es Angular?

Angular es un framework para el desarrollo de aplicaciones web. Está pensado para dividir un proyecto en componentes y ser reutilizadas en proyectos medianos y grandes.

Sus principales competidores son [Vue](#) (proyecto iniciado por Evan You en 2014) y [React](#) (proyecto iniciado por Facebook en 2013)

La última versión (19) introducida a fines de 2024 presenta cambios muy importantes con las 16 versiones anteriores, a tal punto que ha creado un nuevo sitio web para empezar desde cero (sitio presentado a fines de 2023 con la versión 17): [Angular Dev](#)

La versión anterior de [Angular](#) tiene su salida al mercado en 2016 pero además tiene una versión previa no compatible y solo mantenida para proyectos antiguos llamada [Angular.js](#) (2010)

Si tiene que trabajar con proyectos legacy puede tal vez tener la necesidad de ver el [tutorial de Angular previo a la versión 17](#)

El proyecto de Angular es propiedad de la empresa de Google.

La versión de Angular que trabajamos en este tutorial es la 19 (salió el 19/11/2024)

Para desarrollar en forma efectiva una aplicación en Angular debemos instalar al menos dos herramientas básicas:

- Node.js
- Angular CLI (Command Line Interface - Interfaz de línea de comandos)



Instalación de Node.js

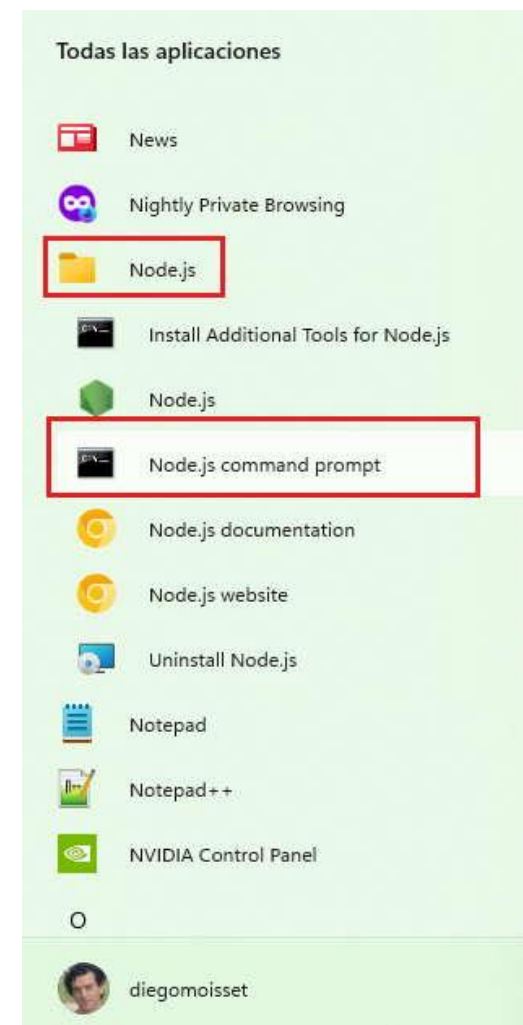
La primer herramienta a instalar será Node.js, esto debido a que gran cantidad de programas para el desarrollo en Angular están implementadas en Node.

Debemos Descargar e instalar la última versión estable de [Node.js](#):

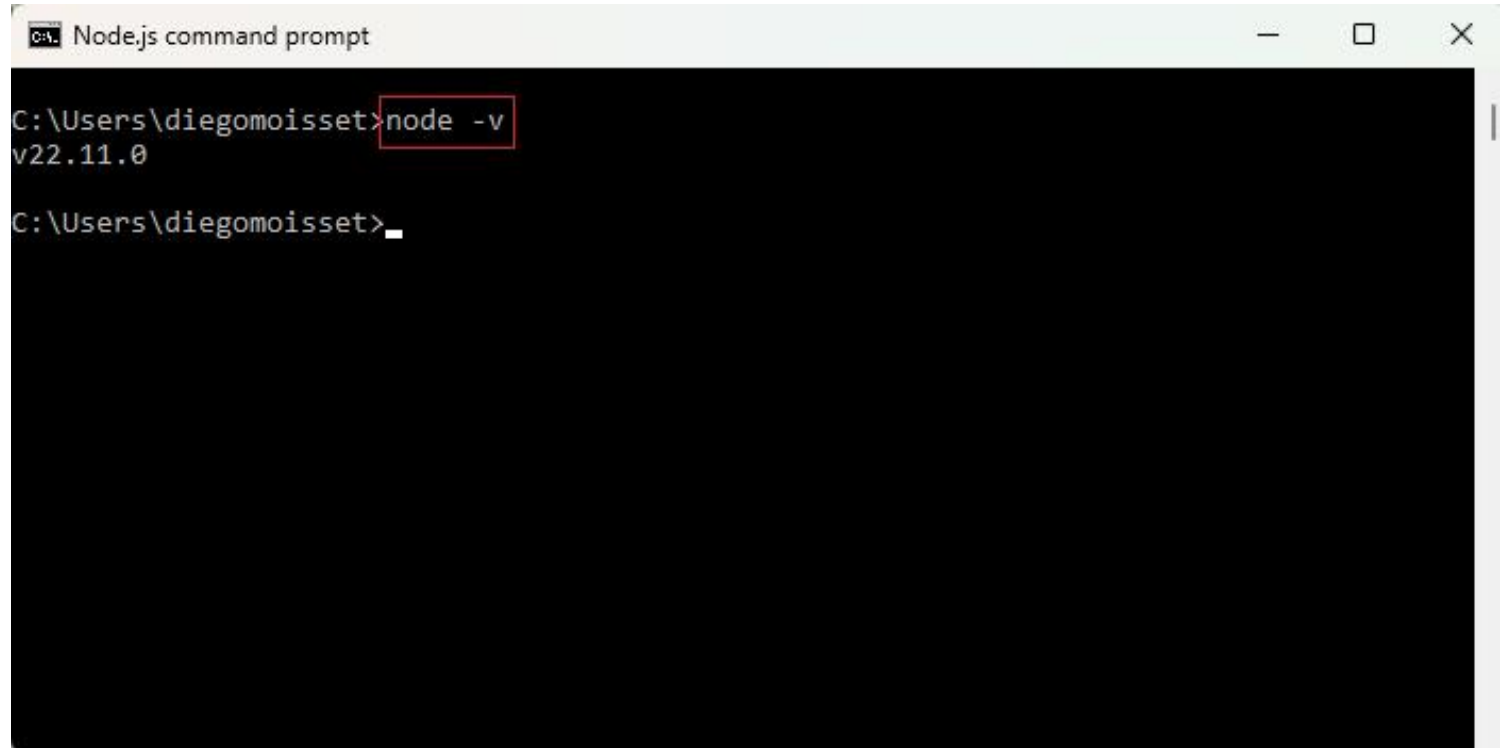




Una vez instalado
debemos ingresar a
la línea de
comandos que nos
provee Nodo.js:



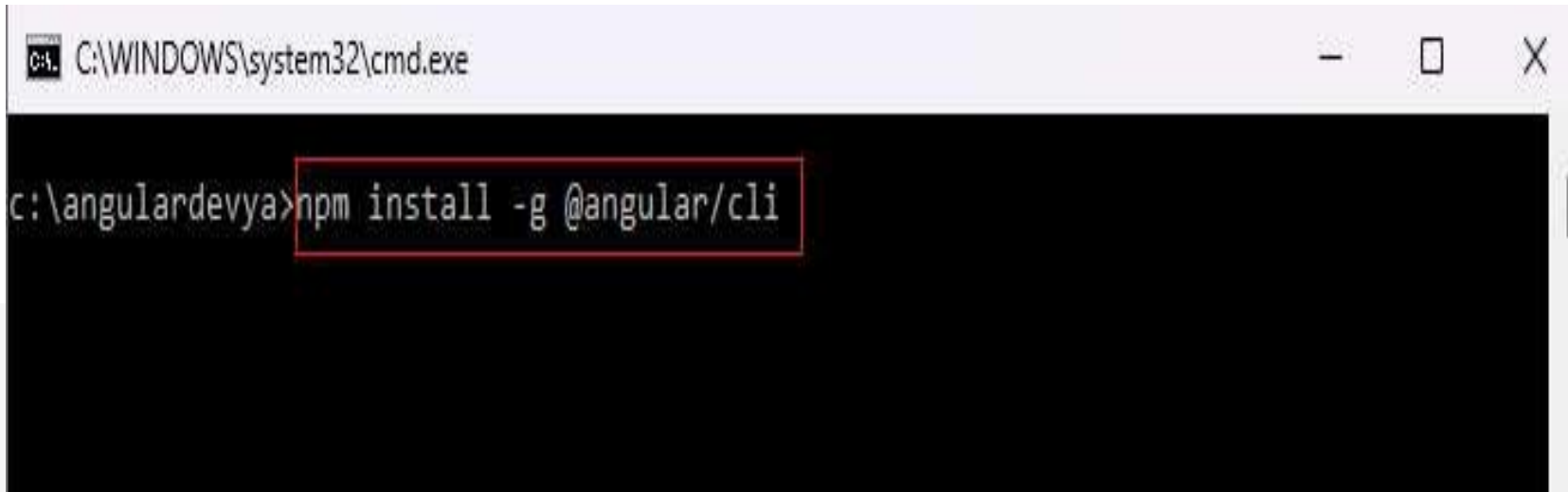
Para comprobar su
correcto funcionamiento
podemos averiguar su
versión:



```
Node.js command prompt
C:\Users\diegomoisset>node -v
v22.11.0
C:\Users\diegomoisset>
```

Instalación de Angular CLI

Para instalar este software lo hacemos desde la misma línea de comandos de Node.js (por eso lo instalamos primero), debemos ejecutar el siguiente comando:



```
C:\WINDOWS\system32\cmd.exe  
c:\angulardevya>npm install -g @angular/cli
```

```
npm install -g @angular/cli
```

Es importante el -g para que se instale en forma global.

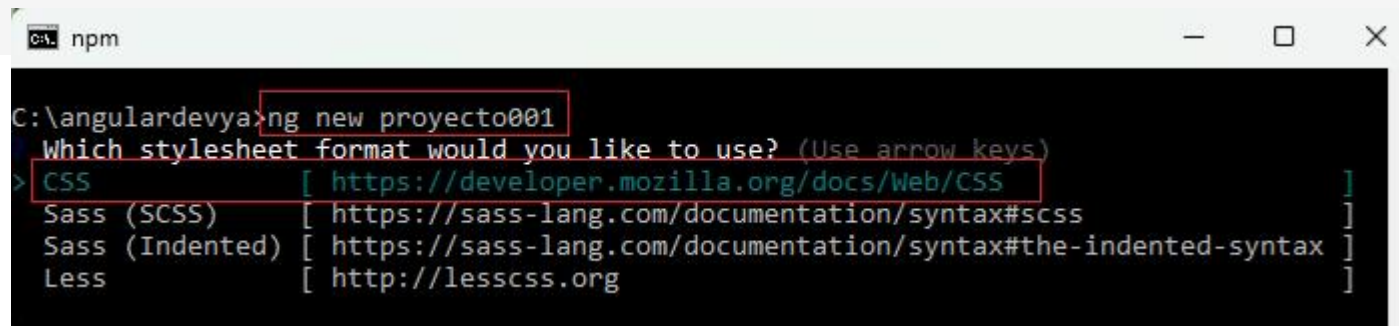
Creación de un proyecto y prueba de su funcionamiento

Para crear un proyecto vamos a utilizar la aplicación Angular CLI que acabamos de instalar en el concepto anterior.

Desde la línea de comandos de Node.js procedemos a ejecutar el siguiente comando:

ng new proyecto001

Seleccionaremos que utilizaremos archivos CSS para los estilos (valor seleccionado por defecto), presionamos la tecla "entrada":



```
C:\angulardevya>ng new proyecto001
Which stylesheet format would you like to use? (Use arrow keys)
> CSS [ https://developer.mozilla.org/docs/Web/CSS ]
  Sass (SCSS) [ https://sass-lang.com/documentation/syntax#scss ]
  Sass (Indented) [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
  Less [ http://lesscss.org ]
```

Seguidamente nos consulta si queremos generar "Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)", por el momento dejamos por defecto que **No** (es una característica agregada a Angular a partir de la versión 17):

```
npm install
C:\angulardevya>ng new proyecto001
✓ Which stylesheet format would you like to use? CSS
https://developer.mozilla.org/docs/Web/CSS
✓ Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? no
CREATE proyecto001/angular.json (2706 bytes)
CREATE proyecto001/package.json (1080 bytes)
CREATE proyecto001/README.md (1533 bytes)
CREATE proyecto001/tsconfig.json (942 bytes)
CREATE proyecto001/.editorconfig (331 bytes)
CREATE proyecto001/.gitignore (629 bytes)
CREATE proyecto001/tsconfig.app.json (439 bytes)
CREATE proyecto001/tsconfig.spec.json (449 bytes)
CREATE proyecto001/.vscode/extensions.json (134 bytes)
CREATE proyecto001/.vscode/launch.json (490 bytes)
CREATE proyecto001/.vscode/tasks.json (980 bytes)
CREATE proyecto001/src/main.ts (256 bytes)
CREATE proyecto001/src/index.html (310 bytes)
CREATE proyecto001/src/styles.css (81 bytes)
CREATE proyecto001/src/app/app.component.html (20239 bytes)
CREATE proyecto001/src/app/app.component.spec.ts (960 bytes)
CREATE proyecto001/src/app/app.component.ts (299 bytes)
CREATE proyecto001/src/app/app.component.css (0 bytes)
CREATE proyecto001/src/app/app.config.ts (318 bytes)
CREATE proyecto001/src/app/app.routes.ts (80 bytes)
CREATE proyecto001/public/favicon.ico (15086 bytes)
| Installing packages (npm)...
```

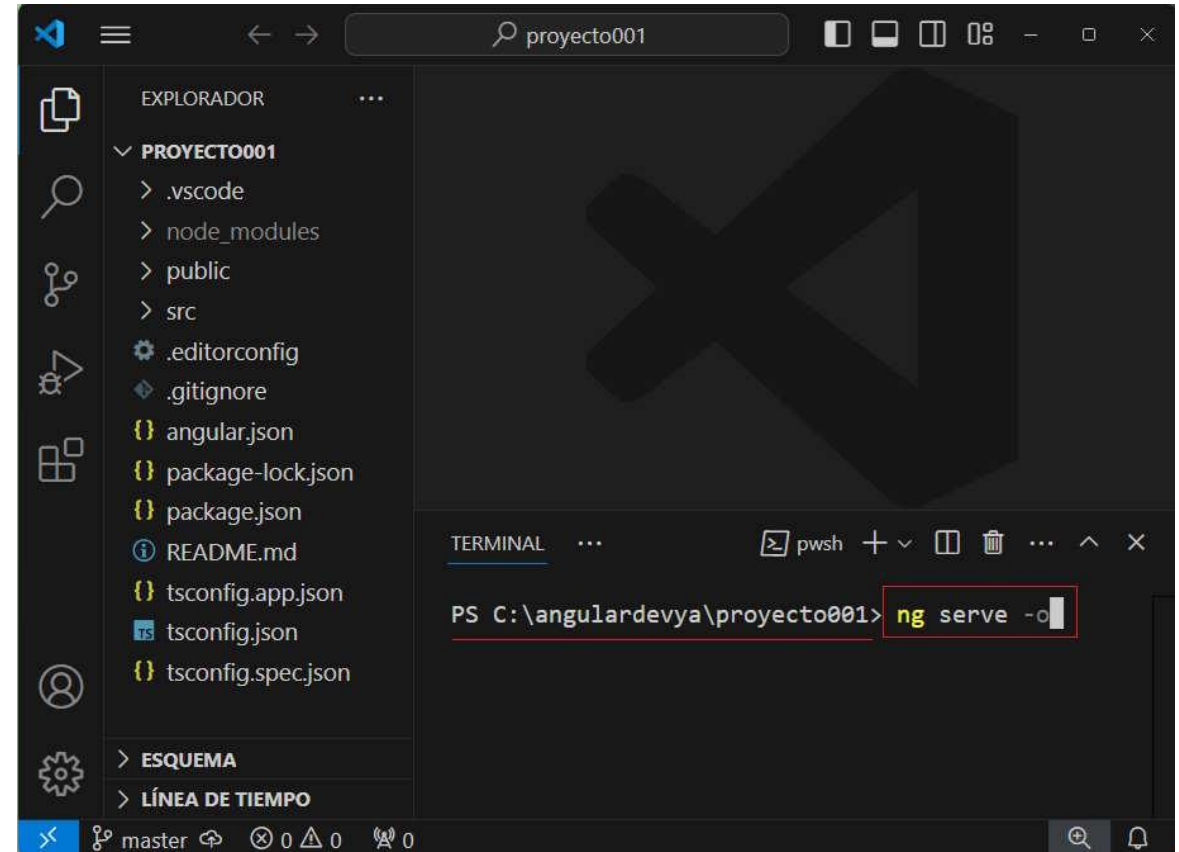


El comando "ng new" crea la carpeta proyecto001 e instala una gran cantidad de herramientas que nos auxiliarán durante el desarrollo del proyecto (255 MB). Como Angular esta pensado para aplicaciones de complejidad media o alta no hay posibilidad de instalar menos herramientas.

El proceso de generar el proyecto lleva bastante tiempo ya que deben descargarse de internet muchas herramientas.

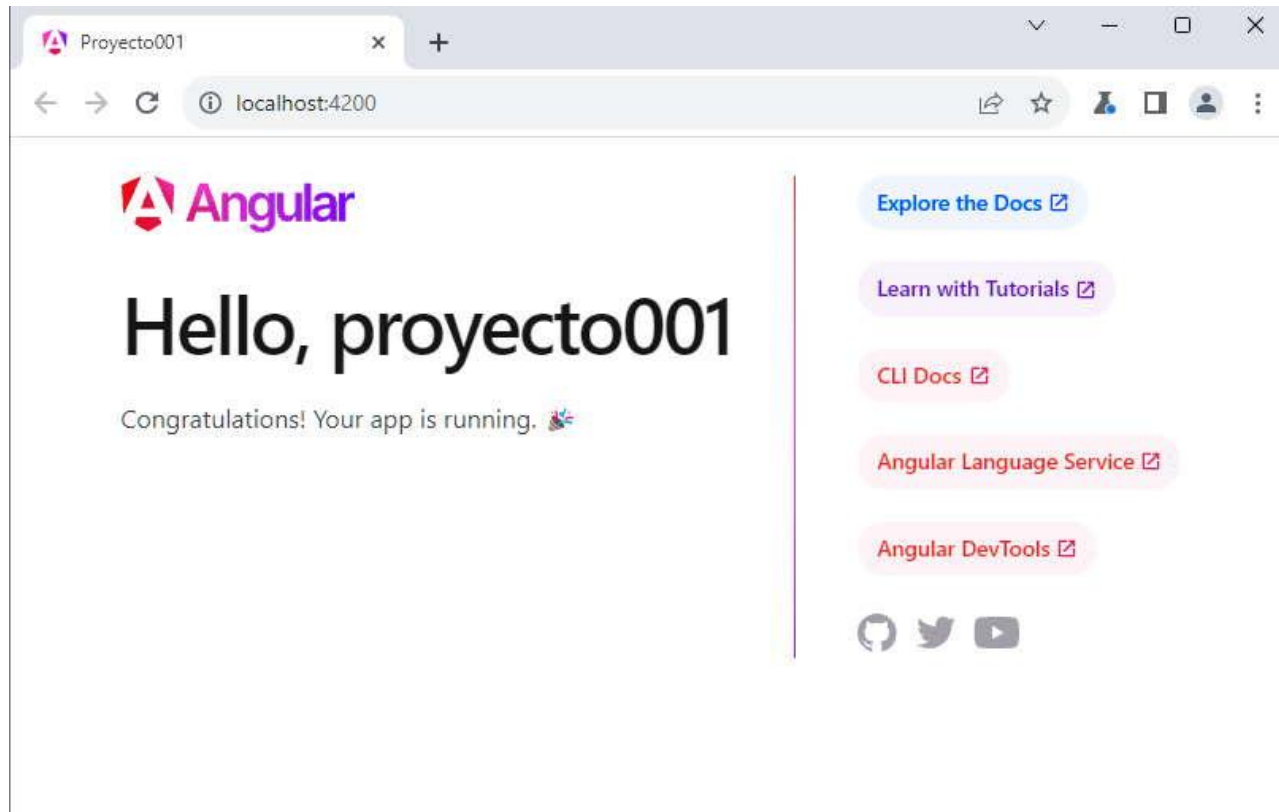
Se genera una aplicación con el esqueleto mínimo, para probarlo podemos abrir el Visual Studio Code y debemos ir a la carpeta que se acaba de crear y lanzar el siguiente comando:

ng serve -o



Este comando arranca un servidor web en forma local y abre el navegador para la ejecución de la aplicación.

En el navegador tenemos como resultado:



En este concepto no me interesa ver todos las carpetas y archivos generados. Solo efectuaremos un cambio para ver como se reflejan en el navegador.

En la carpeta 'proyecto001' hay una subcarpeta llamada 'src' y dentro de esta una llamada 'app', busquemos el archivo 'app.component.html' y procedamos a borrar todo menos la etiqueta <router-outlet />.

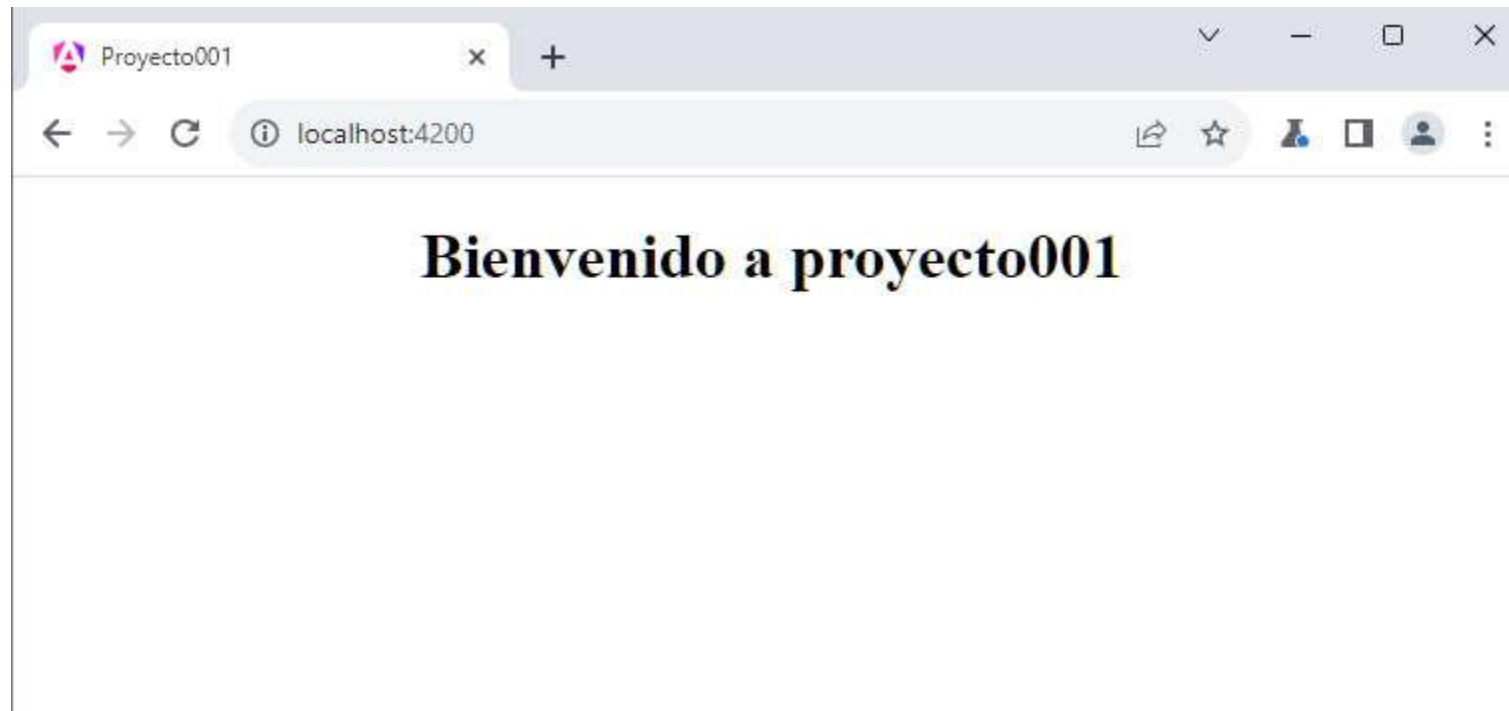
Si leemos las primeras líneas nos informa que siempre debemos modificar el archivo con los algoritmos de nuestro proyecto (se genera a modo de ejemplo):

```
<!-- ***** -->
<!-- ***** The content below ***** -->
<!-- ***** is only a placeholder ***** -->
<!-- ***** and can be replaced. ***** -->
<!-- ***** -->
<!-- ***** Delete the template below ***** -->
<!-- ***** to get started with your project! ***** -->
<!-- ***** -->
```

Disponemos el siguiente código remplazando al generado en forma automática:

```
<h1 style="text-align:center">  
  Bienvenido a {{ title }}  
</h1>  
<router-outlet />
```

Una vez que grabamos los cambios en este archivo podemos ver que automáticamente se ven reflejados en el navegador (recordemos de no cerrar la consola del Visual Studio Code donde ejecutamos el comando "ng serve -o"):



Archivos y carpetas básicas de un proyecto en Angular

Vimos en el concepto anterior que para crear un proyecto en Angular utilizamos la herramienta Angular CLI y desde la línea de comandos escribimos:

```
ng new proyecto001
```

No haremos por el momento un estudio exhaustivo de todos los archivos y carpetas que se crean (más 25.900 archivos y 3300 carpetas en la versión de Angular 19.x), sino de aquellas que se requieren modificar según el concepto que estemos estudiando.

En Angular la pieza fundamental es la 'COMPONENTE'. Debemos pensar siempre que una aplicación se construye a base de un conjunto de componentes (por ejemplo pueden ser componentes: un menú, lista de usuarios, login, tabla de datos, calendario, formulario de búsqueda etc.)

Angular CLI nos crea una única componente llamada 'AppComponent' que se distribuye en 4 archivos:

app.component.ts

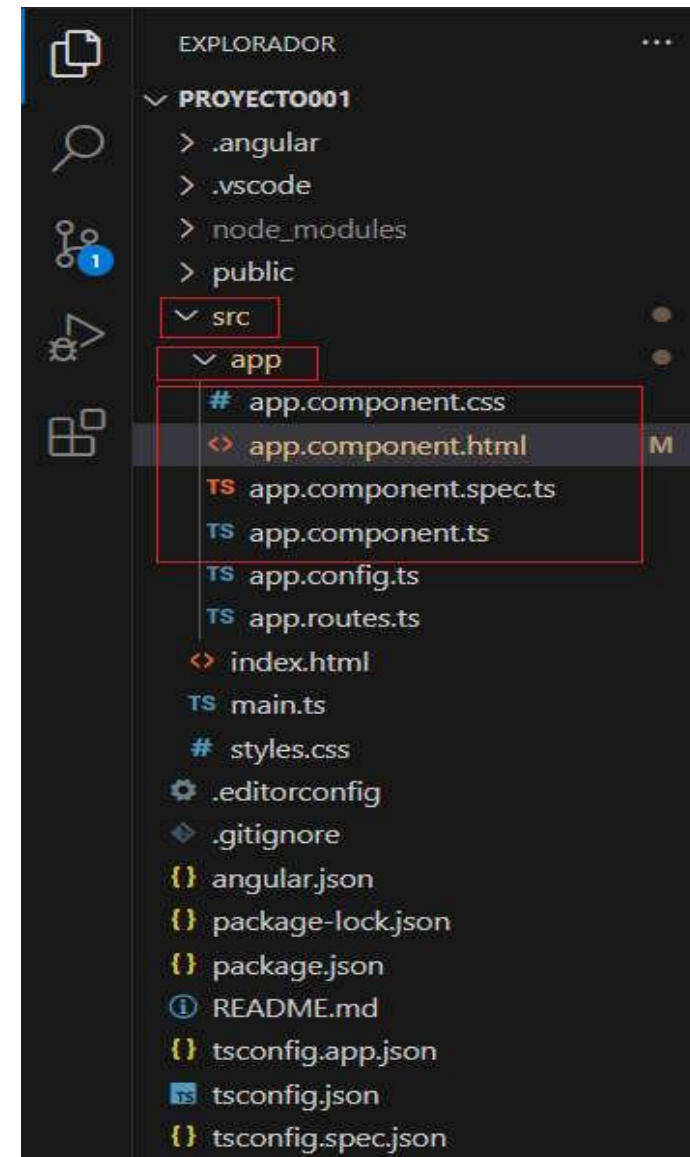
app.component.html

app.component.css

app.component.spec.ts



Todos estos archivos se localizan en la carpeta 'app' y esta carpeta se encuentra dentro de la carpeta 'src':



En Angular se programa utilizando el lenguaje TypeScript que vamos a ir aprendiéndolo a lo largo del curso. El archivo donde se declara la clase AppComponent es 'app.component.ts':

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-root',
  imports: [RouterOutlet],
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'proyecto001';
}
```

La Clase AppComponent define un atributo llamado 'title' y lo inicializa con el string 'proyecto001' que coincide con el nombre del proyecto que creamos:

```
title = 'proyecto001';
```

Dijimos anteriormente que la clase completa se distribuye en otros archivos y podemos ver que mediante la función decoradora @Component le indicamos los otros archivos que pertenecen a esta componente:

```
@Component({  
  selector: 'app-root',  
  imports: [RouterOutlet],  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css']  
})
```

El archivo 'app.component.html' tiene la parte visual de nuestra componente 'AppComponent' y está constituido mayormente por código HTML (cada vez que realicemos un proyecto a este código lo borraremos para resolver nuestro problema, salvo la etiqueta <router-outlet />):

```
<!-- * * * * * -->
<!-- * * * * * The content below * * * * * -->
<!-- * * * * * is only a placeholder * * * * * -->
<!-- * * * * * and can be replaced. * * * * * -->
<!-- * * * * * -->
<!-- * * * * * Delete the template below * * * * * -->
<!-- * * * * * to get started with your project! * * * * * -->
<!-- * * * * * -->

<style>
  :host {
    --bright-blue: oklch(51.01% 0.274 263.83);
    --electric-violet: oklch(53.18% 0.28 296.97);
    --french-violet: oklch(47.66% 0.246 305.88);
    --vivid-pink: oklch(69.02% 0.277 332.77);
    --hot-red: oklch(61.42% 0.238 15.34);
    --orange-red: oklch(63.32% 0.24 31.68);

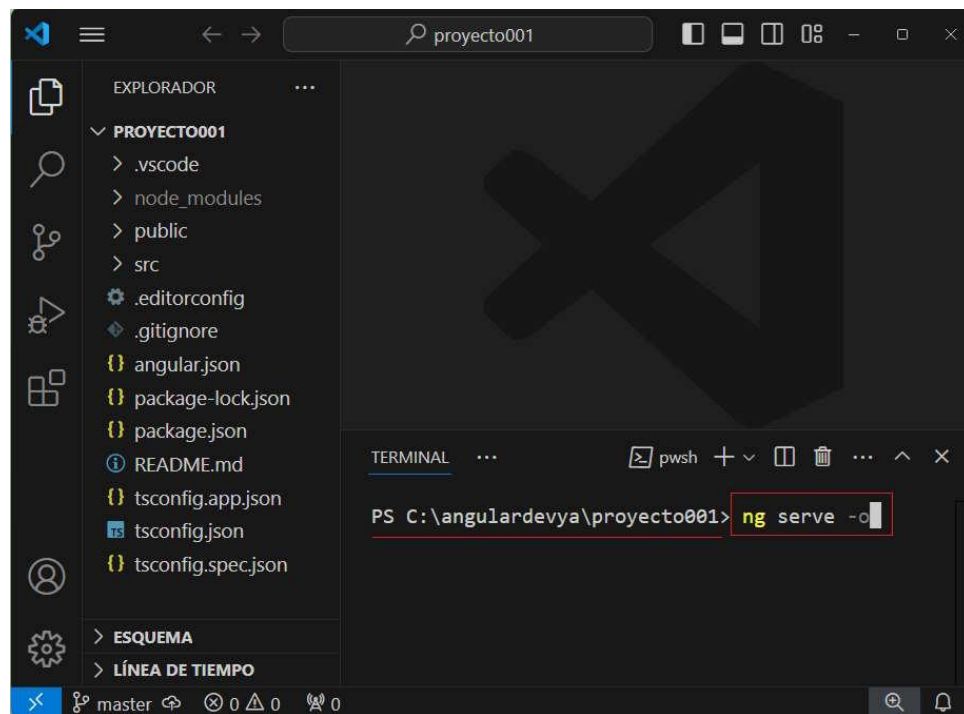
    --gray-900: oklch(19.37% 0.006 300.98);
    --gray-700: oklch(36.98% 0.014 302.71);
    --gray-400: oklch(70.9% 0.015 304.04);

    --red-to-pink-to-purple-vertical-gradient: linear-gradient(
```

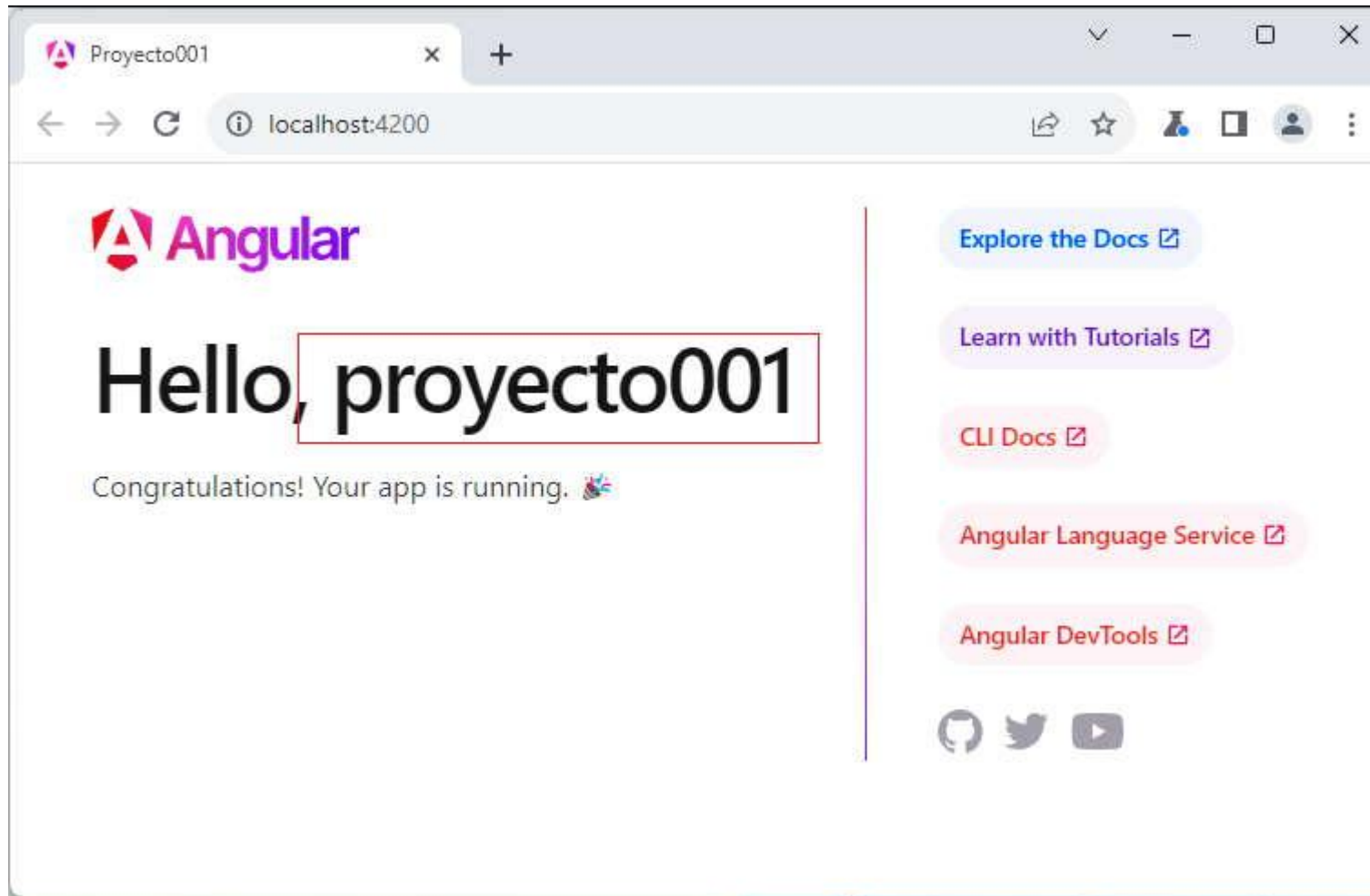
Analizaremos ahora de este trozo de HTML donde aparece el atributo 'title' de la componente:

```
<h1>Hello, {{ title }}</h1>
```

Cuando ejecutamos nuestra aplicación desde la línea de comandos de Node.js:



En el navegador aparece el contenido de la propiedad 'title':



Podemos ver que aparece el string 'proyecto001' y no {{ title }}:

```
title = 'proyecto001';
```

Este concepto de sustitución se llama interpolación y lo veremos en forma más profunda en el concepto siguiente.

Otro archivo que se asocia a la componente 'AppComponent' es 'app.component.css' donde se almacenan todos los estilos que se van a aplicar solo a dicha componente, es decir que quedarán encapsulados en la componente 'AppComponent'.

En la carpeta src del proyecto hay un archivo llamado 'styles.css' donde podemos definir estilos que se aplicarán en forma global a todas las componentes de nuestra aplicación:

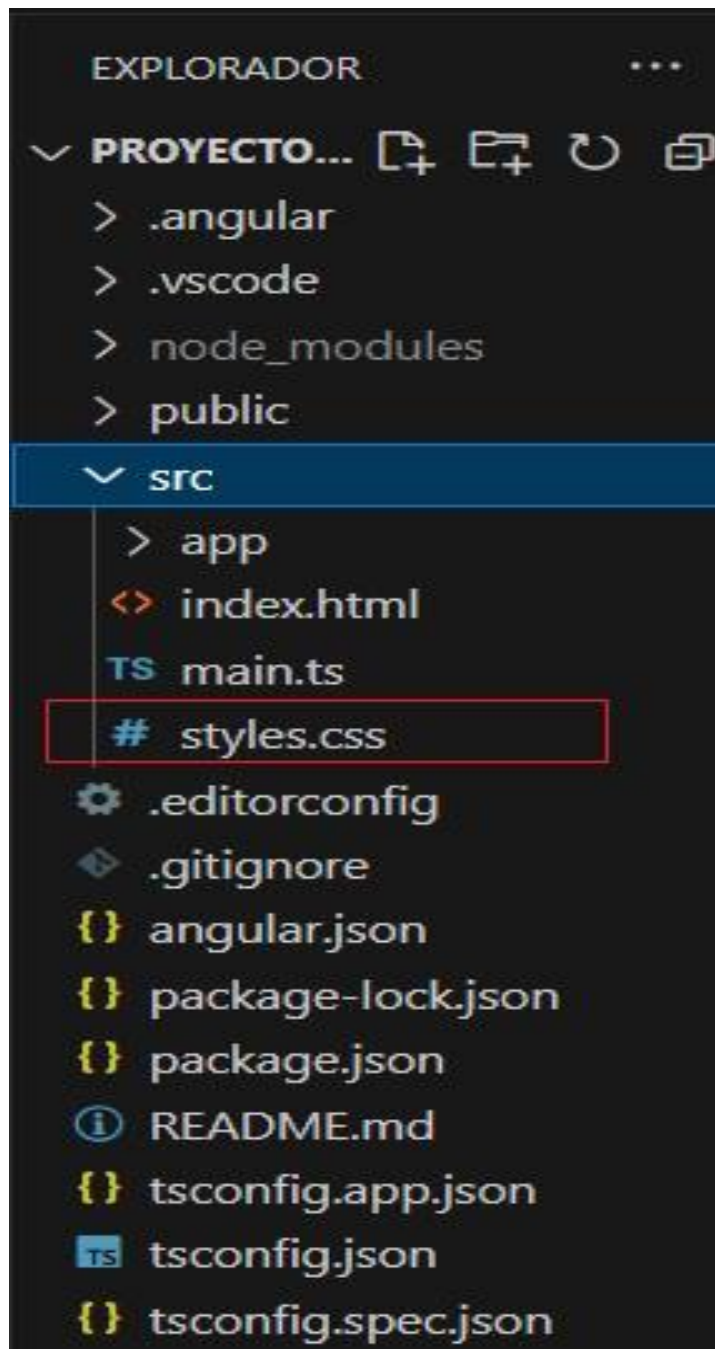


Ya hemos nombrado los tres archivos fundamentales que definen toda componente:

```
app.component.ts
```

```
app.component.html
```

```
app.component.css
```



Queda uno llamado 'app.component.spec.ts' que tiene por objetivo definir código de testing para medir el correcto funcionamiento de la componente (dejaremos para más adelante este concepto)

Podemos acotar que en versiones anteriores a la 17 de Angular se creaba un archivo fundamental de módulos, pero ahora podemos crear aplicaciones sin la obligatoriedad de insertarlos en módulos.

Hay muchos más archivos y carpetas en el proyecto que nos crea Angular CLI pero iremos viendo su objetivo a medida que avancemos en el curso.

Interpolación en los archivos HTML de Angular

Una de las características fundamentales en Angular es separar la vista del modelo de datos. En el modelo de datos tenemos las variables y en la vista implementamos como se muestran dichos datos.

Modificaremos el proyecto001 para ver este concepto de interpolación.

Abriremos el archivo que tiene la clase AppComponent (`app.component.ts`) y lo modificaremos con el siguiente código:

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-root',
  imports: [RouterOutlet],
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  nombre = 'Rodriguez Pablo';
  edad = 40;
  email = 'rpablo@gmail.com';
  sueldos = [1700, 1600, 1900];
  activo = true;

  esActivo() {
    if (this.activo)
      return 'Trabajador Activo';
    else
      return 'Trabajador Inactivo';
  }

  ultimos3Sueldos() {
    let suma = 0;
    for (let x = 0; x < this.sueldos.length; x++)
      suma += this.sueldos[x];
    return suma;
  }
}
```



La clase 'AppComponent' representa los datos de un empleado. Definimos e inicializamos 5 propiedades:

```
nombre = 'Rodriguez Pablo';  
edad = 40;  
email = 'rpablo@gmail.com';  
sueldos = [1700, 1600, 1900];  
activo = true;
```

Definimos dos métodos, en el primero según el valor que almacena la propiedad 'activo' retornamos un string que informa si es un empleado activo o inactivo:

```
esActivo() {  
  if (this.activo)  
    return 'Trabajador Activo';  
  else  
    return 'Trabajador Inactivo';  
}
```

El segundo método retorna la suma de sus últimos 3 meses de trabajo que se almacenan en la propiedad 'sueldos':

```
ultimos3Sueldos() {  
  let suma=0;  
  for(let x=0; x<this.sueldos.length; x++)  
    suma+=this.sueldos[x];  
  return suma;  
}
```

Veamos ahora el archivo html que muestra los datos, esto se encuentra en 'app.component.html':

```
<div>  
  <p>Nombre del Empleado:{{nombre}}</p>  
  <p>Edad:{{edad}}</p>  
  <p>Los últimos tres sueldos son: {{sueldos[0]}}, {{sueldos[1]}} y {{sueldos[2]}}</p>  
  <p>En los últimos 3 meses ha ganado: {{ultimos3Sueldos()}}</p>  
  <p>{{esActivo()}}</p>  
</div>  
<router-outlet />
```


Para acceder a las propiedades del objeto dentro del template del HTML debemos disponer dos llaves abiertas y cerradas y dentro el nombre de la propiedad:

```
<p>Nombre del Empleado:{{nombre}}</p>
```

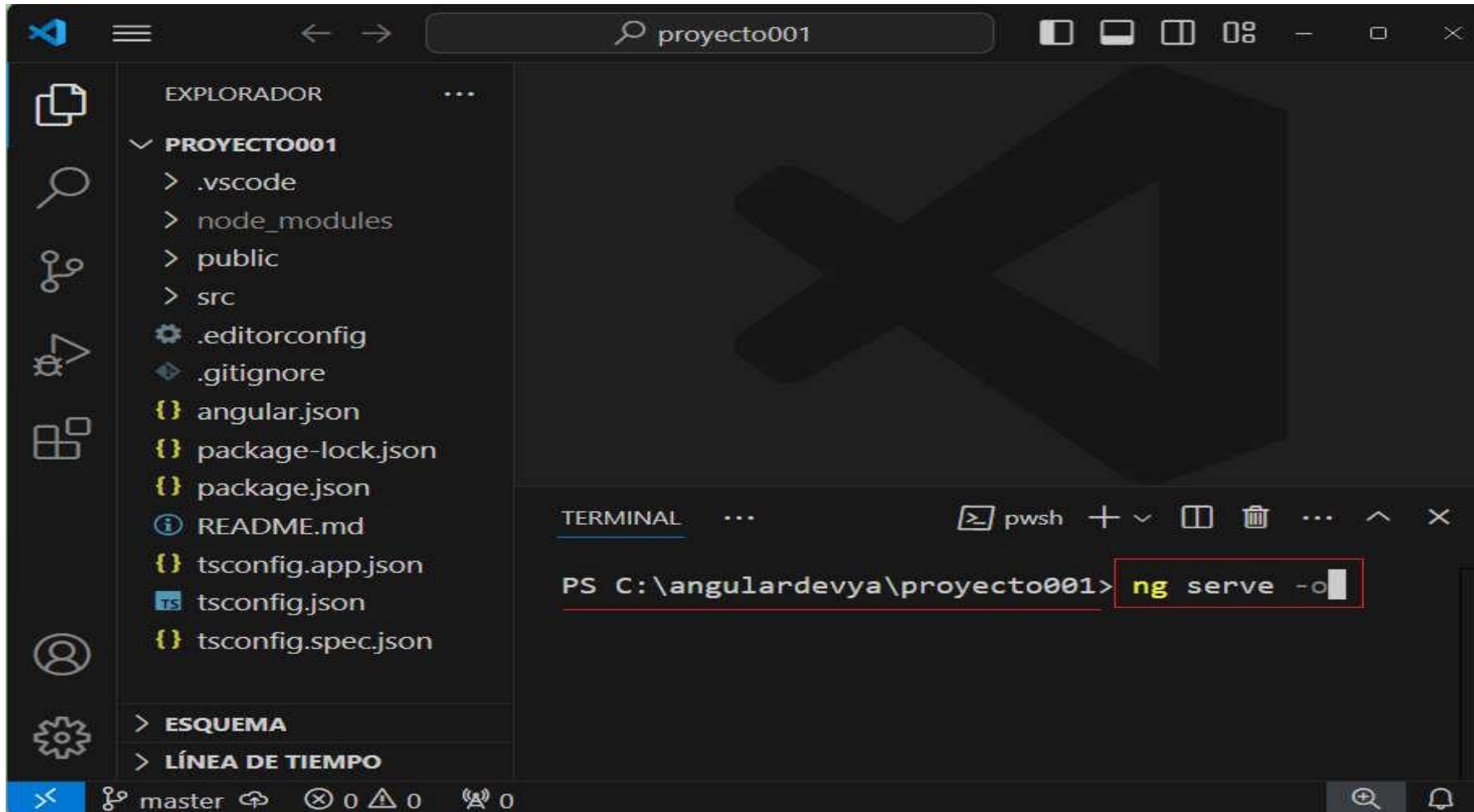
Cuando se tratan de vectores la primer forma que podemos acceder es mediante un subíndice:

```
<p>Los últimos tres sueldos son: {{sueldos[0]}}, {{sueldos[1]}} y {{sueldos[2]}}</p>
```

Finalmente podemos llamar a métodos que tiene por objetivo consultar el valor de propiedades:

```
<p>En los últimos 3 meses ha ganado: {{ultimos3Sueldos()}}</p>  
<p>{{esActivo()}}</p>
```

Cuando ejecutamos nuestra aplicación desde la línea de comandos de Node.js:



En el navegador aparece el contenido de la vista pero con los valores sustituidos donde dispusimos las llaves `{{}}`:



En principio podríamos decir que si los datos son siempre los mismos no tiene sentido definir propiedades en la clase y sustituirlos luego en el HTML, pero luego veremos que las propiedades las vamos a cargar mediante una petición a un servidor web, en esas circunstancias veremos la potencia que tiene modificar las propiedades y luego en forma inmediata se modifica la vista.

Acotaciones

- Dentro de las dos llaves abiertas y cerradas Angular nos permite efectuar una operación:

```
<p>En los últimos 3 meses ha ganado: {{sueldos[0]+sueldos[1]+sueldos[2]}}</p>
```

Primero se opera la expresión dispuesta dentro de las llaves previo a mostrarla.

Otro ejemplo:

```
<p>El empleado dentro de 5 años tendrá:{{edad+5}}</p>
```

- Podemos utilizar la interpolación como valor en propiedades de elementos HTML. Si en la clase tenemos definida la propiedad:

```
sitio='http://www.google.com';
```

Luego en la vista podemos interpolar la propiedad 'url' del elemento 'a' con la siguiente sintaxis:

```
<p>Puede visitar el sitio ingresando <a href="{{sitio}}">aquí</a></p>
```

Sintaxis de Template para
estructuras condicionales y
repetitivas: @if / @else - @for -
@switch/@case/@default

Angular usa `@if` para expresar visualizaciones condicionales en plantillas.

La plantilla o template `@if` nos permiten condicionar si se deben agregar o no bloques de código.

La plantilla `@for` nos permite generar muchos elementos HTML repetidos a partir del recorrido de un arreglo de datos.

Para analizar con un ejemplo estas plantillas procederemos nuevamente a modificar el proyecto001.

En el archivo 'app.component.ts' procedemos a codificar la clase AppComponent con la definición de:

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-root',
  imports: [RouterOutlet],
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  nombre = 'Rodriguez Pablo';

  edad = 40;

  articulos = [{
    codigo: 1,
    descripcion: 'naranjas',
    precio: 540
  }, {
    codigo: 2,
    descripcion: 'manzanas',
    precio: 900
  }, {
    codigo: 3,
    descripcion: 'peras',
    precio: 490
  }];

  generarNumero() {
    return Math.floor(Math.random() * 3) + 1;
  }
}
```



Hemos definido las propiedades nombre, edad y articulos:

```
nombre = 'Rodriguez Pablo';

edad = 40;

articulos = [{
  codigo: 1,
  descripcion: 'naranjas',
  precio: 540
},{
  codigo: 2,
  descripcion: 'manzanas',
  precio: 900
},{
  codigo: 3,
  descripcion: 'peras',
  precio: 490
}];
```


Por otro lado un método que retorna un valor aleatorio comprendido entre 1 y 3:

```
generarNumero() {  
    return Math.floor(Math.random() * 3) + 1;  
}
```

Ahora procedemos a modificar el archivo app.component.html:

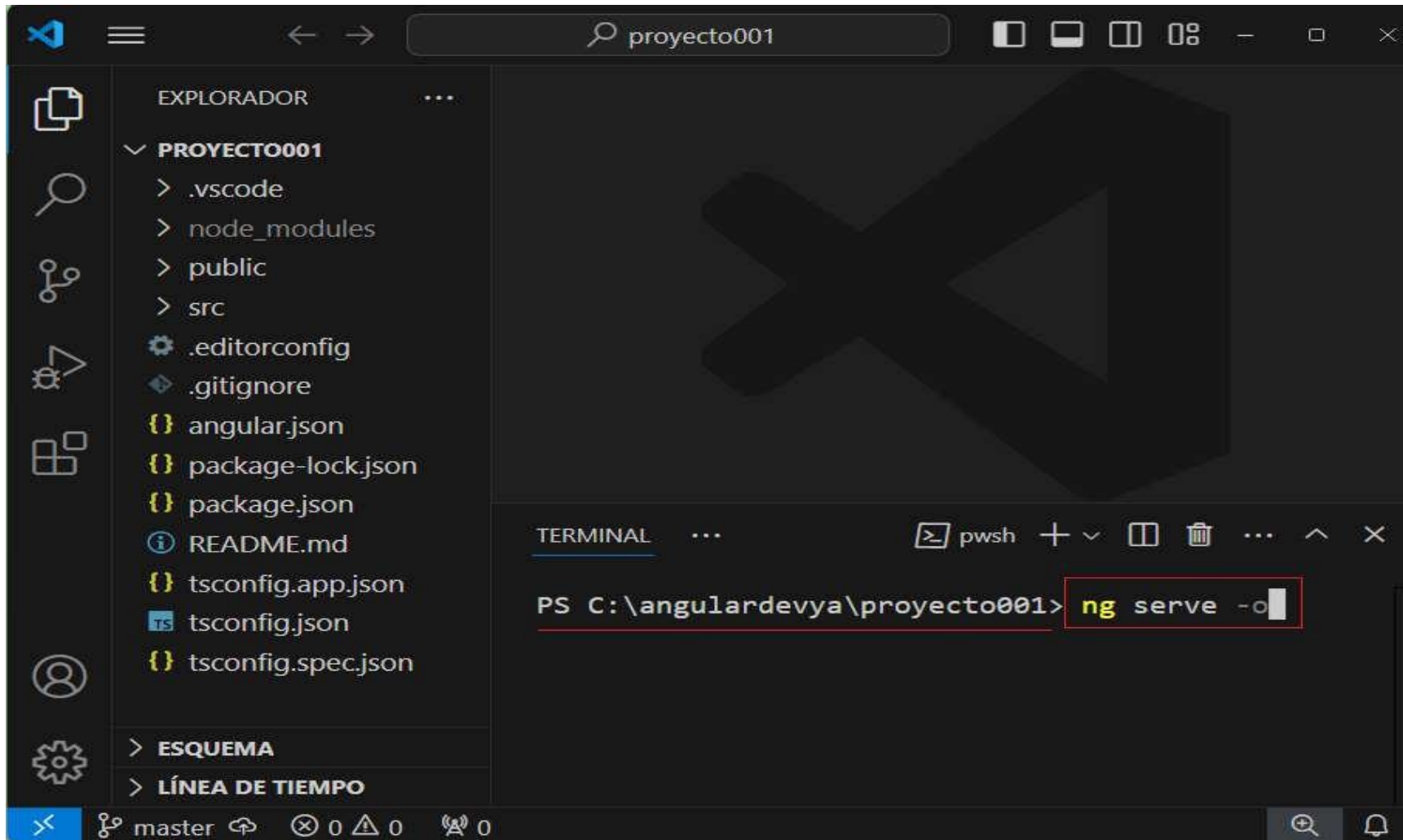


```
<div>
  <h1>Empleado</h1>
  <p>Nombre del Empleado:{{nombre}}</p>
  <p>Edad:{{edad}}</p>
  @if (edad>=18) {
    <p>Es mayor de edad.</p>
  } @else {
    <p>Es menor de edad.</p>
  }
  <h1>Listado de articulos</h1>
  <table>
    @for(articulo of articulos; track articulo.codigo) {
      <tr>
        <td>{{articulo.codigo}}</td>
        <td>{{articulo.descripcion}}</td>
        <td>{{articulo.precio}}</td>
      </tr>
    }
  </table>
```



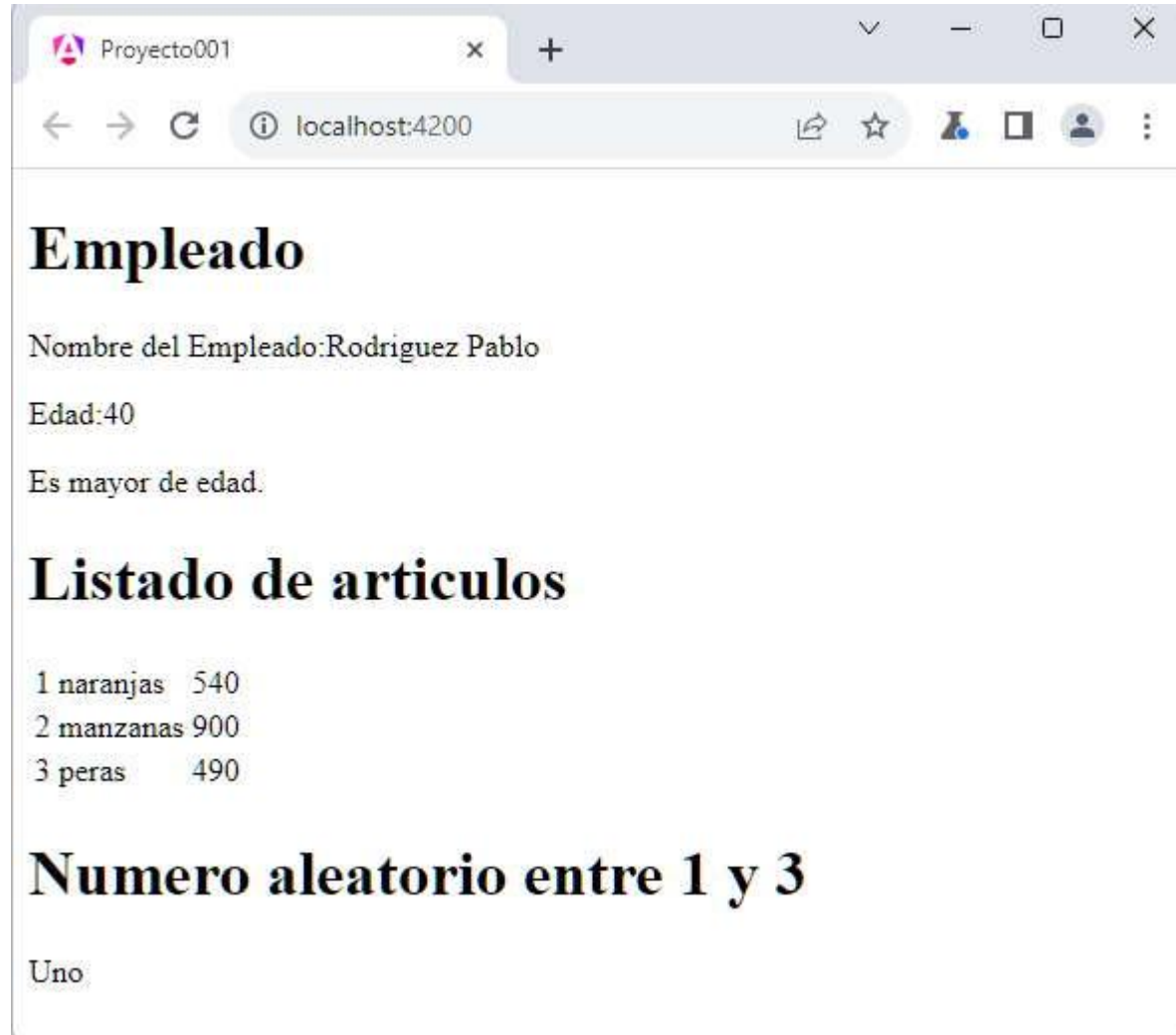
```
<h1>Numero aleatorio entre 1 y 3</h1>
@switch (generarNumero()) {
  @case (1) {
    <p>Uno</p>
  }
  @case (2) {
    <p>Dos</p>
  }
  @case (3) {
    <p>Tres</p>
  }
}
</div>
<router-outlet />
```

Ejecutemos nuestra aplicación desde la línea de comandos de Node.js:





En el navegador aparece el siguiente contenido:



La instrucción @if verifica la condición, en el caso de verificarse verdadero se inserta el bloque contenido entre las llaves:

```
@if (edad>=18) {  
  <p>Es mayor de edad.</p>  
} @else {  
  <p>Es menor de edad.</p>  
}
```

Luego si la condición se verifica falsa se ejecuta el bloque definido en el @else. Podemos probar cambiando la edad por un valor menor a 18.

La instrucción @for nos genera posiblemente muchos elementos HTML repetidos, en este ejemplo una serie de filas de una tabla HTML:

```
@for(articulo of articulos; track articulo.codigo) {  
  <tr>  
    <td>{{articulo.codigo}}</td>  
    <td>{{articulo.descripcion}}</td>  
    <td>{{articulo.precio}}</td>  
  </tr>  
}
```

En cada repetición en la variable 'articulo' se almacena un objeto del arreglo 'articulos'. De esta forma podemos mostrar los datos del objeto respectivo.

Es importante agregar la sentencia track al @for indicando un valor único en cada vuelta del @for (es común utilizar un id o clave)

Por último también disponemos la instrucción @switch donde llamamos al método 'generarNumero' y según el valor retornado entrará en el @case respectivo:

```
@switch (generarNumero()) {  
  @case (1) {  
    <p>Uno</p>  
  }  
  @case (2) {  
    <p>Dos</p>  
  }  
  @case (3) {  
    <p>Tres</p>  
  }  
}
```

Captura de eventos

Otra actividad muy común en una aplicación es la captura de eventos. La presión de un botón, la presión de una tecla, el desplazamiento de la flecha del mouse etc. son eventos que podemos capturar.

El evento más común que podemos encontrar en cualquier aplicación es la presión de un botón. Modificaremos nuevamente el proyecto001 para que la componente AppComponent muestre un etiqueta con un número 0 y luego dos botones que permitan incrementar o decrementar en uno el contenido de la etiqueta.

Nuevamente debemos modificar el archivo 'app.component.ts' con el siguiente código:

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-root',
  imports: [RouterOutlet],
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  contador = 1;

  incrementar() {
    this.contador++;
  }

  decrementar() {
    this.contador--;
  }
}
```

Definimos en la clase la propiedad 'contador' y lo iniciamos con el valor '1':

```
export class AppComponent {  
  contador = 1;  
}
```

Luego otros dos métodos de la clase AppComponent, que serán llamados al presionar alguno de los botones, incrementan en uno o decrementan en uno el valor almacenado en la propiedad contador:

```
incrementar() {  
  this.contador++;  
}  
  
decrementar() {  
  this.contador--;  
}
```

Recordar que las propiedades dentro de los métodos debemos anteceder la palabra clave 'this'

El segundo archivo donde se encuentra la vista de la componente es app.component.html:

```
<div>
  <p>{{contador}}</p>
  <button (click)="incrementar()">Sumar 1</button>
  <button (click)="decrementar()">Restar 1</button>
</div>
<router-outlet />
```

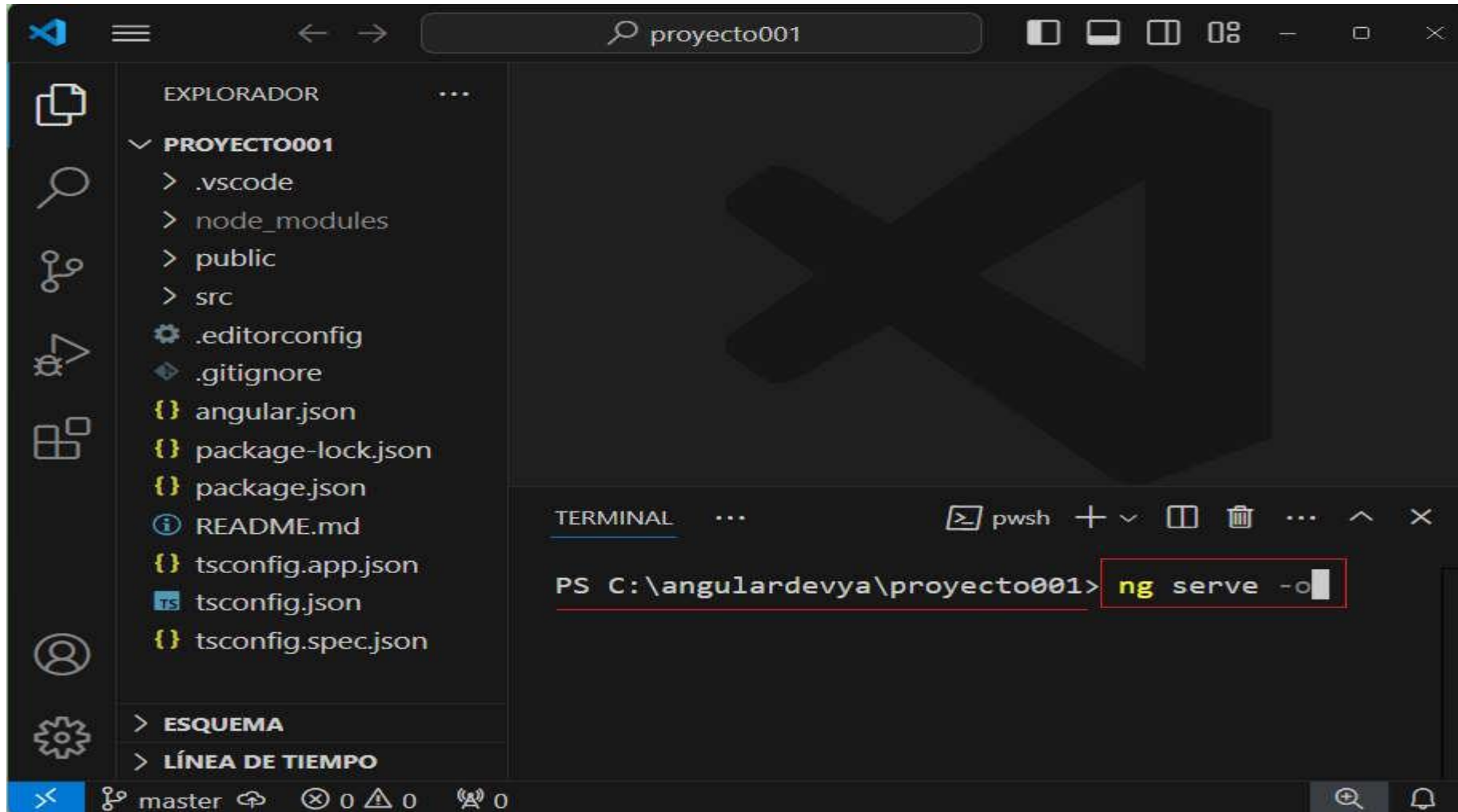
Como ya conocemos mostramos el contenido de la propiedad contador mediante interpolación de string:

```
<p>{{contador}}</p>
```

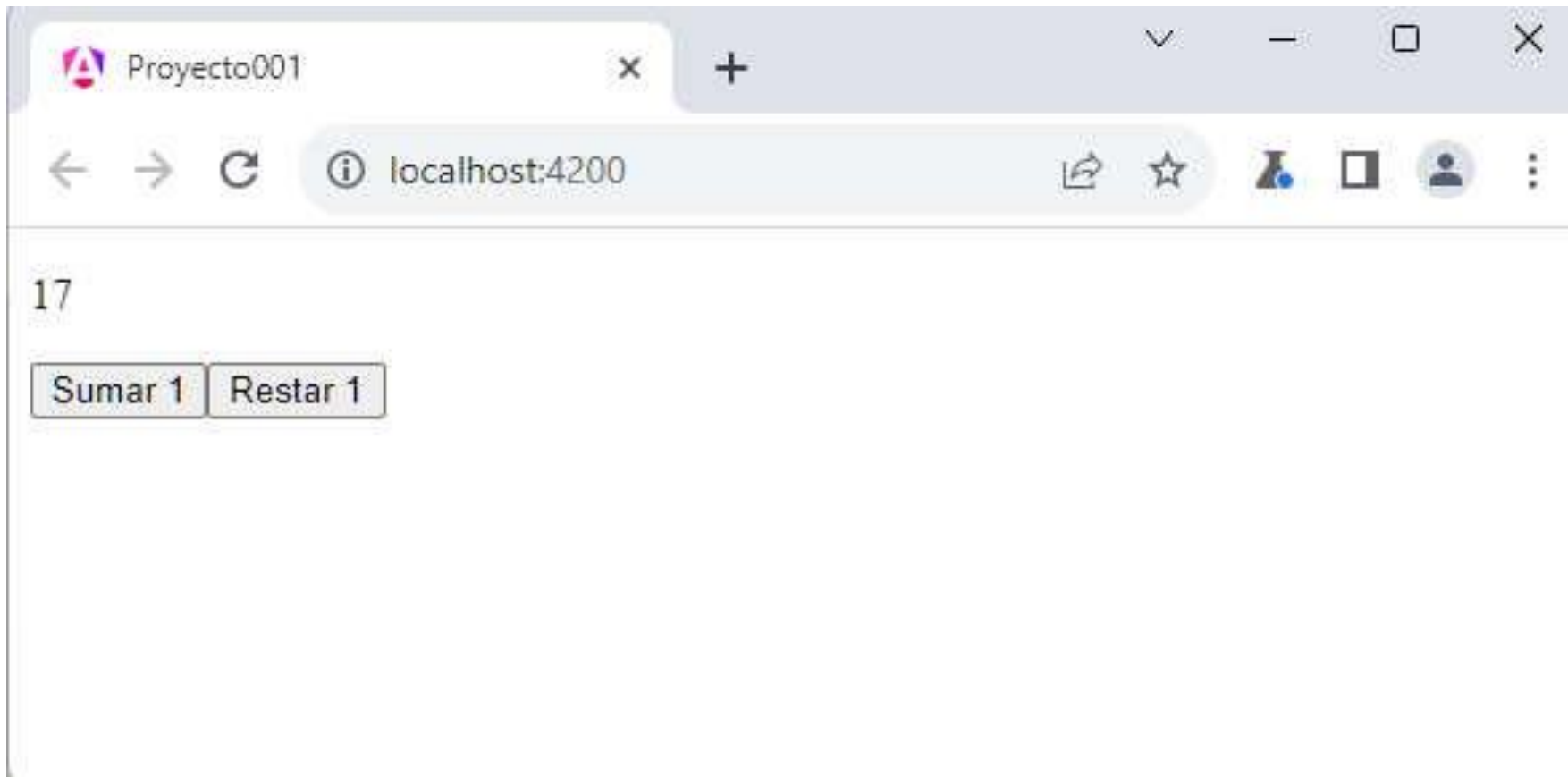
Luego definimos dos elementos HTML de tipo 'button' y definimos los eventos click (deben ir entre paréntesis los nombres de los eventos) y luego entre comillas el nombre del método que se llama:

```
<button (click)="incrementar()">Sumar 1</button>
<button (click)="decrementar()">Restar 1</button>
```

Ejecutemos nuestra aplicación desde la línea de comandos de Node.js:



En el navegador aparece la siguiente interfaz:



Cuando se presiona el botón 'Sumar 1' se llama el método 'incrementar()', en dicho método si recordamos se modifica el contenido de la propiedad 'contador':

```
incrementar() {  
  this.contador++;  
}
```

Lo más importante notar que Angular detecta cuando se modifican valores almacenados en propiedades y automáticamente se encarga de actualizar la interfaz visual sin tener que llamar a algún método.

Este concepto se conoce como 'binding' en una dirección (cambio en atributos de la clase se actualizan en la vista)

Enlace de propiedades (Property Binding)

El property binding es una herramienta poderosa en Angular que nos permite mantener sincronizadas las propiedades de una componente con las propiedades de elementos del DOM, facilitando así la creación de aplicaciones dinámicas e interactivas.

Vimos que para la captura de eventos en los elementos HTML disponemos entre paréntesis el evento y asociamos un método, para el enlace de propiedades debemos disponer entre corchetes el nombre de la propiedad y le asignamos un atributo definido en la clase.

Modifiquemos nuevamente el archivo 'app.component.ts' con el siguiente código:

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';

@Component({
  selector: 'app-root',
  imports: [RouterOutlet],
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  nombre = ''

  fijarNombre1() {
    this.nombre = 'Juan';
  }

  fijarNombre2() {
    this.nombre = 'Ana';
  }
}
```

Definimos un atributo llamado nombre, el cual se modifica en dos métodos.

El segundo archivo donde se encuentra la vista de la componente y donde definimos el property binding es app.component.html:

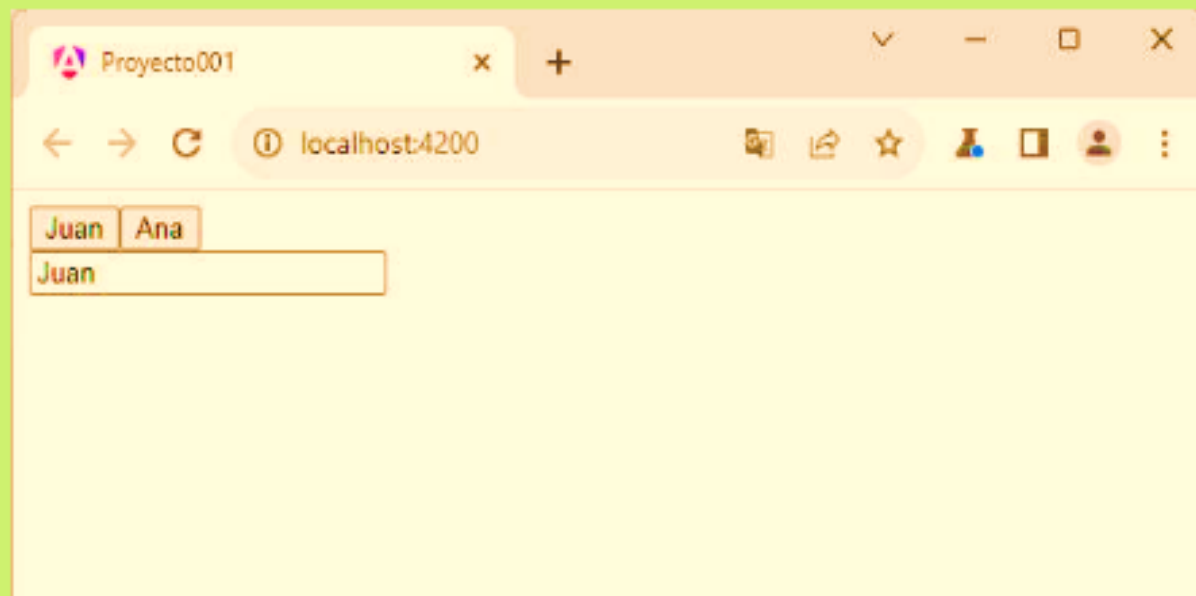
```
<div>
  <input type="button" (click)="fijarNombre1()" value="Juan">
  <input type="button" (click)="fijarNombre2()" value="Ana">
</div>
<input type="text" [value]="nombre">
<router-outlet />
```

Debemos disponer entre corchetes el nombre de la propiedad y asignarle el nombre del atributo definido en la clase:

```
<input type="text" [value]="nombre">
```

Cuando el operador presione cualquiera de los dos botones, al modificarse el atributo "nombre" en la clase, automáticamente se actualiza la vista con el valor asignado al atributo.

Si ejecutamos la aplicación y presionamos el primer botón:



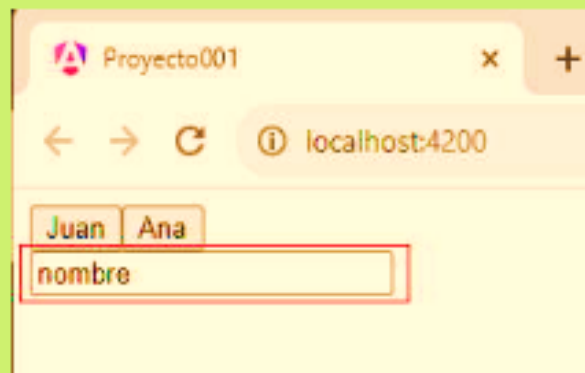
Tener cuidado con la importancia de los corchetes que envuelven la propiedad:

```
<input type="text" [value]="nombre">
```

Si no disponemos los corchetes, recordemos que el navegador va a mostrar la cadena "nombre" dentro del control de entrada de datos:

```
<input type="text" value="nombre">
```

Y tenemos como resultado en la página:



Muchas Gracias